

Programmieren von Klassen in Java

- Abstrakte Klassen
- Schnittstellen

30.11.2016


Monika Tepfenhart

1

Abstrakte Klassen

- Eine **abstrakte Klasse** ist eine Klasse, von der keine Instanzen erzeugt werden können. Sie fasst Gemeinsamkeiten zusammen und gibt eine Schnittstelle vor, die von ihren abgeleiteten Klassen erfüllt werden muss

Schlüsselwort zur Kennzeichnung von abstrakten Klassen



```
public abstract class Artikel {  
    ...  
}
```

30.11.2016

Monika Tepfenhart

2

Abstrakte Klassen

- Eine **abstrakte Methode** hat eine Signatur, aber keinen Methodenrumpf. Sie gibt vor, welche Funktionalität in Unterklassen implementiert werden

Klasse muss abstrakt sein, da sie eine abstrakte Methode deklariert

```
public abstract class Artikel {  
    ...  
    public abstract String getTwitterBeschreibung();  
}
```

Kennzeichnung der abstrakten Methode

Abstrakte Methoden definieren keinen Methodenrumpf

30.11.2016

Monika Tepfenhart

3

Abstrakte Klassen

- In einer Unterklasse von „Artikel“ muss diese Methode implementiert werden

Implementierung der abstrakten Methode in der Unterklasse Buch

```
public class Buch extends Artikel {  
    ...  
    public String getTwitterBeschreibung() {  
        return "Buch: '" + titel + "' von " + autor;  
    }  
}
```

30.11.2016

Monika Tepfenhart

4

Abstrakte Klassen

- Das Implementieren einer geerbten abstrakten Methode in einer Unterklasse wird vom Compiler erzwungen
 - Die einzige Möglichkeit, eine abstrakte Methode nicht in einer direkten Unterklasse zu implementieren, ist, sie dort wieder als abstract zu kennzeichnen
 - Das hat allerdings zur Folge, dass diese Unterklasse zu einer abstrakten Klasse wird und so die Verantwortung der Implementierung an weitere Unterklassen delegiert wird
-

30.11.2016

Monika Tepfenhart

5

Abstrakte Klassen

- Es ist möglich, Variablen vom Typ einer abstrakten Klasse zu deklarieren
- Da aber keine Objekte von dieser abstrakten Klasse erzeugt werden können, muss der zugewiesene Wert der so deklarierten Variable zwingend ein Objekt einer abgeleiteten Klasse sein

30.11.2016

Monika Tepfenhart

6

Abstrakte Klassen

```
public static void main(String[] args) {  
    Artikel artikel = new Artikel();  
  
    Artikel artikel;  
  
    artikel = new Buch();  
  
    System.out.println(  
        artikel.getTwitterBeschreibung());  
}
```

Instanziierung nicht erlaubt, da Artikel eine abstrakte Klasse ist

Eine Variable kann vom Typ einer abstrakten Klasse deklariert werden.

Zuweisung ist auf Grund von Zuweisungskompatibilität erlaubt

Für den Aufruf wird die Implementierung aus der Klasse Buch gewählt.

30.11.2016

Monika Tepfenhart

7

Schnittstellen

- Ein **Interface** definiert eine Menge von Methoden, die von Klassen implementiert werden können
- Weil die Methoden erst in den Klassen realisiert werden, findet man in Interfaces nur Methoden ohne einen Rumpf

30.11.2016

Monika Tepfenhart

8

Schnittstellen

- **Schnittstellen** werden in Java mit dem Schlüsselwort `interface` definiert. Die Syntax erinnert sehr an die einer Klasse, mit dem Unterschied, dass alle Methoden abstrakt sind.
- Die Methoden bestehen lediglich aus der Signatur und haben somit keinen durch geschweifte Klammern eingeschlossenen Rumpf
- Auf diese Weise können Sie spezifizieren, was eine implementierende Klasse können muss, ohne das „Wie?“ vorwegzunehmen

30.11.2016

Monika Tepfenhart

9

Schnittstellen

- Bei der Definition von Schnittstellen ist zu beachten, dass sie grundsätzlich zustandslos sein müssen. Es ist daher bis auf Konstanten nicht erlaubt, Attribute zu definieren
- Mithilfe des Schlüsselwortes *implements* kann eine Klasse anzeigen, welche Schnittstelle(n) sie realisiert. Damit verpflichtet sie sich, für jede Methode dieser Schnittstelle(n) eine Implementierung anzubieten

30.11.2016

Monika Tepfenhart

10

Schnittstellen

The diagram illustrates the relationship between a Java interface and its implementation. It consists of two code blocks. The first block defines a public interface named `EineSchnittstelle` with a single method `public void eineMethode(int param);`. An arrow points from the text "Name der Schnittstelle in der Definition und in der Implementierung" to the interface name. The second block shows a public class `ImplementierendeKlasse` that implements `EineSchnittstelle`. It contains the implementation of `eineMethode` and other methods indicated by ellipses. Annotations include: "Methoden müssen implementiert werden" pointing to the method signature in the class; "hier wird die Schnittstelle implementiert" pointing to the `implements` keyword; and "es folgen ggfs. weitere Methoden" pointing to the ellipses. A small button labeled "Rechteckiges Ausschneiden" is visible in the top right of the second code block.

```
public interface EineSchnittstelle {  
    public void eineMethode(int param);  
}  
  
public class ImplementierendeKlasse implements EineSchnittstelle  
{  
    ...  
    public void eineMethode(int param){  
        ...  
    }  
    ...  
}
```

30.11.2016

Monika Tepfenhart

11

Schnittstellen

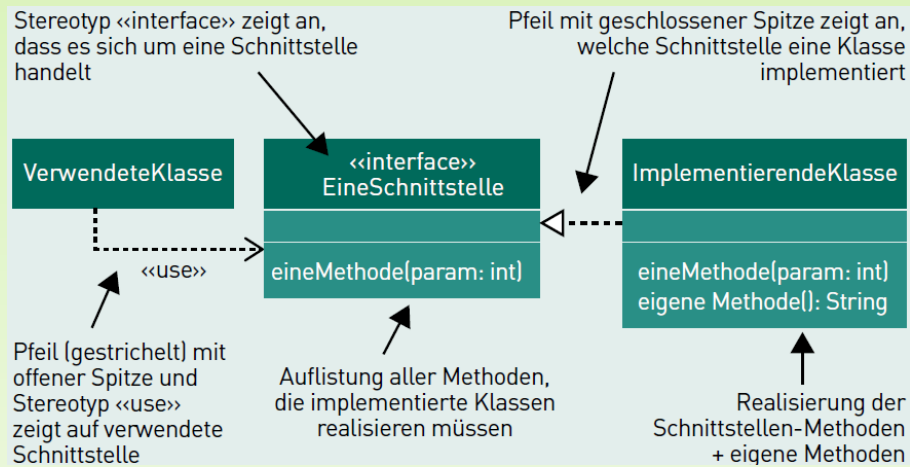
- In UML ähnelt die Notation von Schnittstellen der von Klassen
- Zur Unterscheidung wird das Stereotyp `<<interface>>` verwendet
- Mit einer *use-Assoziation*, dargestellt durch einen gestrichelten Pfeil mit offener Spitze und Stereotyp `<<use>>` wird gekennzeichnet, dass eine Klasse eine Schnittstelle verwendet. D.h., die Klasse ruft mindestens eine Methode der Schnittstelle auf

30.11.2016

Monika Tepfenhart

12

Schnittstellen



30.11.2016

Monika Tepfenhart

13

Schnittstellen

- Die implementierende Klasse zeigt, ähnlich zur Vererbungs-Notation, mit einer geschlossenen Pfeilspitze auf die Schnittstelle. Im Gegensatz zur Vererbung wird der Pfeil aber gestrichelt dargestellt

30.11.2016

Monika Tepfenhart

14

Schnittstellen

- Das Interface definiert eine Reihe von Methoden, die unabhängig von der Implementierung für die dauerhafte Speicherung von Kundendaten notwendig sind. Die Kundenverwaltung benutzt dieses Interface in einer ihrer Methoden, spielt also die Rolle der verwendenden Klasse

30.11.2016

Monika Tepfenhart

15

Schnittstellen

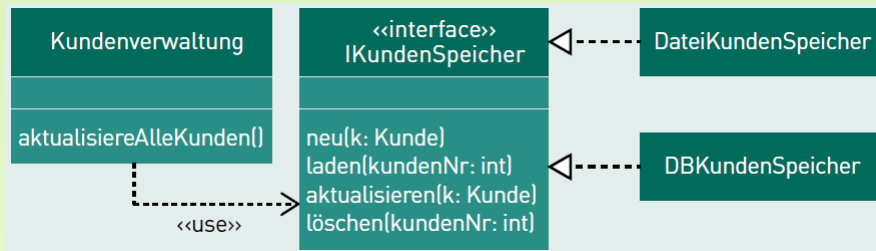
- Die Alternativen der Schnittstellen-Implementierung (DateiKundenSpeicher und DBKundenSpeicher) spielen die Rolle der implementierenden Klassen. Sie bieten je eine Realisierung zu jeder Methode, die in der Schnittstelle IKundenSpeicher vereinbart wurde

30.11.2016

Monika Tepfenhart

16

Schnittstellen



30.11.2016

Monika Tepfenhart

17

Schnittstellen

- Einen zum Klassendiagramm passenden Java-Code finden Sie auf der nächsten Seite.
 - Sie zeigt die Verwendung der Schnittstelle an einer Beispiel-Methode, die Definition des Interfaces sowie eine Beispiel-Implementierung.
 - (Die Anweisungen im Rumpf der implementierten Methoden sind aus Platzgründen abgeschnitten.)

30.11.2016

Monika Tepfenhart

18

Schnittstellen

Schnittstelle mit vereinbarten Methoden

```
public interface IKundenSpeicher {  
    public void neu(Kunde k);  
    public Kunde laden(int kundenNr);  
    public void aktualisieren(Kunde k);  
    public void löschen(int kundenNr);  
}
```

Implementierung der Schnittstellen-Methoden

```
public class DBKundenSpeicher implements IKundenSpeicher{  
    public void neu(Kunde k) { ... }  
    public Kunde laden(int kundenNr) { ... }  
    public void aktualisieren(Kunde k) { ... }  
    public void löschen(int kundenNr) { ... }  
}
```

30.11.2016

Monika Tepfenhart

19

Schnittstellen

Verwenden von DBKundenSpeicher als konkrete Implementierung der Schnittstelle IKundenSpeicher

```
public class Kundenverwaltung {  
    private Kunde k1;  
    private Kunde k2;  
    ...  
    private IKundenSpeicher kundenSpeicher = new DBKundenSpeicher();  
  
    public void aktualisiereAlleKunden(){  
        kundenSpeicher.aktualisieren(k1);  
        kundenSpeicher.aktualisieren(k2);  
        ...  
    }  
}
```

Aufruf einer Schnittstellen-Methode

30.11.2016

Monika Tepfenhart

20

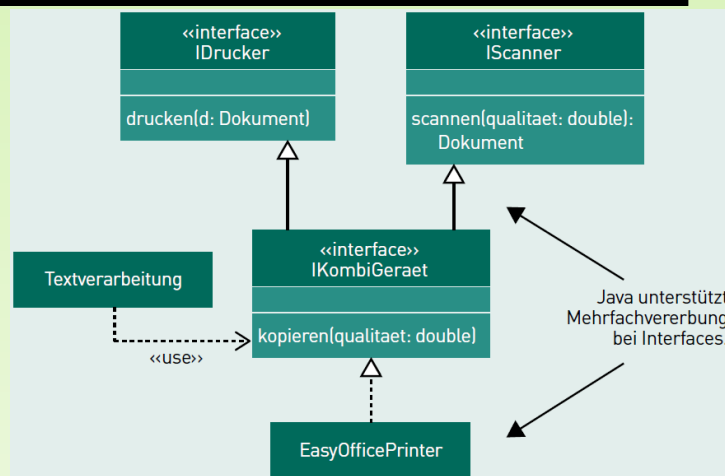
Schnittstellen

- Weil die benötigten Methoden zur Speicherung von Kundendaten nun in einer Schnittstelle definiert sind, können sie ohne großen Aufwand beliebig ausgetauscht werden (Flexibilität)
- Umgekehrt kann die Implementierung zur Speicherung von Kundendaten prinzipiell an jeder weiteren Programmstelle eingesetzt werden, an der ein solches Interface benötigt wird (Wiederverwendbarkeit)

Schnittstellen

- Weil Interfaces auch von anderen Interfaces erben können. Auf diese Weise können Gemeinsamkeiten von Schnittstellen zusammengefasst und je nach Verwendungszweck ohne großen Wartungsaufwand erweitert werden. Bezogen auf Interfaces unterstützt Java außerdem das Konzept der Mehrfachvererbung. Das heißt, im Gegensatz zu Klassen können Interfaces sogar von mehr als einem Interface erben. Trotzdem wird mit extends das gleiche Schlüsselwort verwendet

Schnittstellen



30.11.2016

Monika Tepfenhart

23

Schnittstellen

```
public interface IKombiGeraet extends IDrucker, IScanner {
    public void kopieren(double qualitaet);
}

public interface IDrucker {
    public void drucken(Dokument d);
}

public interface IScanner {
    public Dokument scannen(double qualitaet);
}

public class EasyOfficePrinter implements IKombiGeraet {
    public void kopieren(double qualitaet) { ... }
    public void drucken(Dokument d) { ... }
    public Dokument scannen(double qualitaet) { ... }
}
```

30.11.2016

Monika Tepfenhart

24

Schnittstellen

- Parallelen - Interfaces und abstrakte Klassen:
 - Abstrakte Klassen können mit abstrakter Methoden Klassen-übergreifende Funktionalitäten festzulegen
 - Bei Interfaces spielt die Vererbungshierarchie keine Rolle. Auch „unverwandte“ Klassen können einem gleichen Interface zugeordnet werden
 - Eine Klasse kann mehr als ein Interface implementieren
 - Bei Interfaces ist es verboten, implementierte und abstrakte Methoden zu vermischen. Ebenso wird durch das Verbot Attribute zu definieren verhindert, zustandsbehaftete Interfaces zu programmieren
-